Doc:

Issue: A

Date: 05.07.2016

**Electronics R&D**

**FPGA Design Rules**

**(FOR INTERNAL USE)**

|  |  |
| --- | --- |
| Author  (date/signature) | IKA |
| Checked by  (date/signature) |  |
| Checked by  (date/signature) |  |
| Approved by  (date/signature) |  |

*Document3*

|  |  |  |  |
| --- | --- | --- | --- |
| **DISTRIBUTION LIST** | | | |
| **Nº of copies** | **Company** | **Department** | **Name** |
|  | ETEL-only | Electronics R&D |  |

|  |  |  |
| --- | --- | --- |
| **MODIFICATIONS** | | |
| **Issue Rev.** | **Date** | **Modified pages** |
| A | 05.07.2016 | Initial release |

**TABLE OF CONTENTS**

[1. DOCUMENT PURPOSE 1](#_Toc447175019)

[2. DOCUMENTS 1](#_Toc447175020)

[2.1. Applicable documents 1](#_Toc447175021)

[2.2. Referenced documents 1](#_Toc447175022)

[3. Conclusion 2](#_Toc447175023)

FIGURES

TABLES

# DOCUMENT PURPOSE

The purpose of this document is to list all requirements to the design process of FPGAs done within ETEL’s Electronics R&D Department.

# DOCUMENTS

## Applicable documents

## Referenced documents

# Abbreviations

CC ClearCase®

SCM Software Configuration Management

# General Requirements

The following requirements are imposed on the FPGA design process:

* All FPGA source code and related documentation shall be version controlled. The enterprise-level SCM tool is ClearCase. All final releases (and especially releases distributed externally) have to be version-controlled under ClearCase. Development, research, proof-of-concept work is allowed to be version-controlled in alternative SCM tools, like GIT.
* All releases have to be traced, i.e. supplied with enough unique identification attributes. They also have to be appropriately labeled in the SCM tool. The goal is to be able to fully and faithfully obtain any given version of the codebase and to recreate any given FPGA release. The latter usually means that the FPGA compilation tools shall also be version controlled.
* The version numbering has to follow the established ETEL standard.

# Definitions

FPGA codebase

This is the set of source files used as input to the FPGA compilation process

FPGA source files

These include all RTL files (e.g. VHDL or Verilog text files, precompiled EDIF and NGC files), as well as all files (project files, makefiles, configuration option files, etc.) describing the project and driving its compilation.

FPGA release

The output of the FPGA compilation process, usually a result of a synthesis-P&R-assembly cycle. Includes the FPGA bitstream(s), together with all compilation reports produced by the tools.

# Release Contents

Each new release differs from the previous one via one or more changes (change items) that can be divided into three classes:

* Bugfixes (BUG)
* New features (NEW)
* Updates to already existing features (UPD)

This classification is frequently fuzzy, as many change items have the characteristics of several or all of these change classes. If this is the case, the change item shall be named preferably going top-down on the above list.

If the SCM allows it, it is recommended that each separate change item be checked-in as a self-sufficient changeset.

# Release Preparation Process

The release preparation process shall contain the following steps:

1. In case no previous release of the same version line exists (e.g. 3.11 is the version line for 3.11A, 3.11B1, 3.11beta, etc.), an appropriate ROOT label (see below) is created and applied to the correct place in the “/main” branch.
2. A new CC view is established having an appropriately named release branch as the integration branch (see below). This release branch stems from the ROOT label defined in the previous step.
3. Release work is performed on a private branch stemming from the release branch (see below).
4. When the release work is finished, the private branch is merged onto the release branch. This includes all generated files.
5. All checked out files are checked in.
6. A release label is created and applied to the just merged versions on the release branch.
7. The FPGA release checklist is to be completed.
8. Document the release as described below.

# Release Compilation

The compilation process uses the compilation tools (e.g. Altera Quartus® or Xilinx Vivado®) to produce output files (like bitstreams and compilation reports) from the input source files.

Schematically this process can be seen in Figure 1.

RTL files

RTL configuration files

Output files (bitstreams & compilation reports)

Release ID files (timestamp)

Tool invocation command line

Tool configuration files and settings (variables & options)

**Elements that shall be version-controlled**

Figure 1: FPGA compilation process and its artefacts

# Release Quality and Destination

According to their purpose each release can be in the following “quality” state:

* Development/research/proof-of-concept – this is a release meant for everyday development work
* Release candidate – this is a release line with frozen features state, which is iteratively being promoted via successful validation steps into an official release. Release candidates can be subdivided into the following classes:
  + Alpha-release – meant for testing by the department or by internal clients
  + Beta-release – a functional release that can be given to external clients for further testing and for feedback on the features and their functionality
  + Release-Candidate (infrequently used) – a completely feature-frozen version, that is still undergoing final validation at ETEL, but can be sent to clients to enable their early adoption efforts.
* Official release – this is a production-quality release that has successfully passed all required validation tests

According to the destination each release can be classified as:

* Internal – normally meant exclusively for ETEL-internal needs. Can be eventually given to an external client only with the written explicit permission of the department Director.
* External – can be given to an external client.

# Release Identification

Releases are uniquely identified by:

* A version
* A time-of-compilation timestamp
* An identifier of the codebase version (e.g. a CC label, or GIT commit hash)

According to the general ETEL rules, FPGA releases are numbered using several alphanumeric components combined in the following way:

*<major version – 1 digit>.<minor version – 2 digits><bugfix capital letter><client-specific version – 1 digit><release quality indicator>*

The meaning of the components is as follows:

* Major Version­ (e.g. 1, 2, 3) [required] – identifies a version that contains a considerable backwards incompatibility, e.g. one that applies to a new HW revision, or that requires considerable changes to the interfacing SW.
* Minor Version (e.g. 00, 01, 12) [required] – identifies a version that does not present a drastic break to the backwards compatibility. It can require, however, some changes to the interfacing SW.
* Bugfix capital letter (e.g. A, B, C) [required] – identifies a bugfix to a precedent
* Client-specific or configuration-variant version (e.g. 1, 2, 3) [optional, if not present assumed to be 0] – identifies a version containing a client-specific feature set or a specifically configured feature set.
* Release-quality indicator (e.g. x, x1, x2, alpha1, alpha2, beta1, rc1, rc2)

Successive versions shall be presented with lexicographically increasing string values, however there is no requirement to use successive numbers.

# Version Control with ClearCase®

The following rules guide the version control of the FPGA codebase:

* There is a **“/main”** branch, containing all source codebase developed until that time. The contents of this branch shall be compilable (eventually given a set of certain configuration options, e.g. for excluding or including certain features) and shall have passed all validation (this includes unit, integration, system and regression) tests defined at the moment of the merge to this branch.
* No development work shall take place on the **“/main”** branch. Code to this branch shall be brought only via merging from release, development or bugfix branches (after having passed all validation tests).
* All release work is done on a separate branch, which is a direct descendant of **“/main”** and is called “main-ETL\_BR\_<product>\_V<release version line>. E.g. main-ETL\_BR\_ULTIMET\_PCI\_FPGA\_V1.13. This branch, once created, cannot be deleted. This branch is created off from the **“/main”** branch at a point labeled by a label like ETL\_LB\_<product>\_ROOT\_V<release\_version\_line> (e.g. ETL\_LB\_ULTIMET\_PCI\_FPGA\_ROOT\_V1.13)
* After the codebase for a release is frozen, it shall be labeled by a *release label* like ETL\_LB\_<product>\_REL\_V<release\_version> (e.g ETL\_LB\_ULTIMET\_PCI\_FPGA\_REL\_V1.13A, or ETL\_LB\_ULTIMET\_PCI\_FPGA\_REL\_V1.13beta1x).
* The release label has to be applied to the following files and folders:
  + All source files used as input for the FPGA compilation
  + The produced bitstream(s) and any auxiliary output files that serve to identify the compiled release (like time\_stamp.vhd). In case any of these output files is copied over to another location in the file system (e.g. into the sw\_fw/xw\_fpga folder), then the copies shall also receive the same label.
* In case the released codebase receives changes (e.g. due to fixes of issues found during validation), the release label shall be moved to the changed version of the affected source files.
* Bug fixes performed on a release branch shall be merged to the upstream **“/main”**. New features and updates may also be merged to the upstream. If needed, change items can be backported to previous versions, by merging onto the respective previous-version branch.
* The merge history of any change item performed on a release branch shall be followed. In this way, it can be easily traced if a bugfix done for a certain release is also integrated into another one.
* During the work on a release, any changes performed on the upstream **“/main”** branch or any other release branch, and deemed necessary, can be merged onto the current release branch.
* The release branch is an integration branch, i.e. all checkouts of files on this branch has to be done on a private branch. The latter can be a generic development branch (e.g. ETL\_BR\_ULTIMET\_TCPIP\_FPGA\_DEV\_IKA, or ETL\_BR\_ULTIMET\_FPGA\_DEV\_IKA, or ETL\_BR\_ULTIMET\_DEV\_IKA, or ETL\_BR\_DEV\_IKA), a generic debug branch (e.g. or ETL\_BR\_ULTIMET\_TCPIP\_DBG\_AMATBUG\_IKA), or a feature-specific one (e.g. ETL\_BR\_ ULTIMET\_FPGA\_FEA\_TRANSNETTIMER\_IKA)
* A CC config\_spec for working with a release branch shall use the following template:

# Show all checkedout files  
element \* CHECKEDOUT

# Show files belonging to the private branch that stems from the release one

element \* …/<rel-branch>/<priv\_branch>/LATEST

# Create a private branch stemming from the release one

mkbranch <priv\_branch>

element \* …/<rel-branch>/LATEST

end mkbranch

# Create a release branch stemming from the release ROOT

# Newly created files are directly branched to the release branch

mkbranch <rel\_branch>

# make a release branch for yet untouched files

element \* <rel\_root\_label>

# if ever a new file has to be added

element \* /main/LATEST

end mkbranch

----------  
Note:   
<rel\_branch> shall be replaced with the real release branch name, e.g. main-ETL\_BR\_ULTIMET\_PCI\_FPGA\_V1.13  
<priv\_branch> shall be replaced with the real private branch name, e.g. ETL\_BR\_DEV\_IKA  
<rel\_root\_label> shall be replaced with the real root label name, e.g. ETL\_LB\_ULTIMET\_PCI\_FPGA\_ROOT\_V1.16

* Note that files that have been newly created while working on a release branch will not have the release root label!
* When checking in files the following comments shall be provided:
  + For source files:

<blank line>  
- CR-<call-ID from TIS>: <Description of the change>  
================================================================================ (80x)

* + For output files (e.g. rbf, time\_stamp.vhd, etc.)

FPGA ver.: <fpga version, e.g. 1.13A1>  
Timestamp: <timestamp as seen in time\_stamp.vhd, e.g. 1345678912>  
CC Label: <name of CC label used to mark the source files>

* When merging, the following comments shall be provided:
  + In the merge dialog: no comment is needed since we don’t know where it is used
  + When checking in the merged version:

<blank line>  
- Merge from <branch\_name/version>  
================================================================================ (80x)

# Documenting FPGA Releases

Each FPGA release is to be documented in the following way:

* In the RELEASE\_NOTES.txt files (on branch **“/main”** !)
* In the Excel file UltimET Light FPGA change history.xslx
* In TIS with the Call-ID that has been opened to perform the changes
* In the ETEL product history database O:\doc\_int\common\qa\suivi\_produits\_mcd.accdb where a new record is open for each TIS Call-ID and for each separate FPGA.

# Miscellaneous (to sort)

FPGA design consists (build of materials) of:

Source files - RTL (VHDL, Verilog), compiled netlists (e.g. NGC), IPs, etc.

Platform project files (e.g. Vivado, Quartus)

Build results - bitstreams, reports

Simulation files - testbenches, stimulation vectors, simulation results

Is CR approved and validated?

Make a list of all FPGA source files that change

To reproduce a given compilation we need:

the original versions of the source files (and intermediated derived objects)

the original version of the code tools

optionally, the original version of the OS and environment where the tools were run

the settings (variable and options) and build rules used for building each target

After a compilation, the following files must be preserved:

linker output files

all tool compilation reports

Process

Identify all places that need to be changed

During changing, append a small comment like -- [IKA] CR-103869 at the end of each line. They can be deleted after all changes are working

Make sure there is a label/baseline identifying the state of the source files BEFORE changing any of them.

Perform the changes on a bugfix or a feature branch

After development has finished and after first testing, merge the bugfix/feature branch onto its parent

All bugfix or feature dev has to be done with a CR!

General development, concept demo, research trials, etc. can be done without a CR on a pure DEV branch!